



Developing Dashboard Widgets

Tiger introduces a new feature called Dashboard, which provides a new and unique class of mini-applications called Widgets. While Dashboard will be a powerful feature for Tiger users, for developers it introduces a rich new development environment. Widgets are quick to develop and easy to deploy, and they can leverage all of Tiger's advanced technologies. Widgets are perfect for working with small amounts of data or interacting with other applications, both on your desktop and across the web. And they provide an excellent way to add functionality to an existing application. This means there are many opportunities for developers, whether creating new products, or adding market-differentiating value to an existing one.

Based on Web Kit technologies, Dashboard Widgets are created using a mix of HTML, JavaScript, and CSS. This extends the ability to develop Widgets to a very wide audience. If you know how to create a web page, then you know how to create a capable Widget. But Widgets aren't limited by their use of web-based technologies. They can tap into the immense power of Mac OS X. Better yet, you can create great Widgets in hours or days, instead of months or years. And, for ADC Members with access to pre-release versions of Tiger, there's a Dashboard SDK that helps you get up to speed as quickly as possible. (ADC Members with active software seed keys can find the Dashboard SDK in the "Downloads: Developer Tools" section of the ADC Member Site.)

This article shows you how Dashboard Widgets work, and provides an introduction to how to develop and deliver modular functionality in bite-sized pieces.

Understanding Widgets

Widgets are ready to use when Dashboard is activated and disappear when the Dashboard is put away, allowing quick "at-a-glance" usage. Since Dashboard is part of the system and doesn't require the installation of any additional components, Widgets can become essential tools for users. Each Widget is task-specific and lightweight. And if a Widget needs a little bit more interface room for setting preferences, Dashboard lets them "flip" over and use the back of the Widget. (For more on how Dashboard works from a user's perspective, see the Dashboard "Sneak Preview" page.)

Widgets fall into one of the following three categories:

- **Accessory Widgets** are self-contained and don't require support from an application or Internet access. Clocks, timers, calculators, and note-takers fall into this category.
- **Application Widgets** are associated with a full-fledged application. This kind of Widget enhances the application by providing a less complicated and often read-only interface. The iTunes Controller and Address Book Widgets fall into this category.
- **Information Widgets** are designed to work with data from the Internet. These Widgets allow you to monitor external events such as the weather, flight status, or stock prices.

Get the Tiger Early Start Kit for Developers



Tiger Articles

Mac OS X v.10.4 Tiger Overview: New features, APIs, and frameworks.

Working with Spotlight: Powerful search technologies.

Developing Dashboard Widgets: Quick and easy mini-applications.

Developing 64-bit Applications: Break the 32-bit limitations.

Working with Automator: Streamline everyday manual tasks.

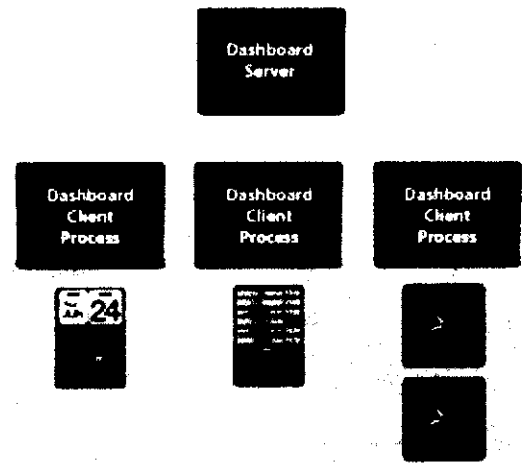
Working with Xcode 2.0: Compelling new features in app development.



Dashboard Architecture

The runtime architecture of Dashboard consists of the following components:

- **The Dashboard server**, a lightweight process that manages the Dashboard user interface including the Widget bar, close box, and Widget launch effects.
- **Dashboard client processes**, providing all the needed glue between the Dashboard server and individual Widgets as well as the Web Kit view for the Widget to display its user interface within. The Dashboard server launches one client process per running Widget.
- **The Widget instances**, displaying data to, and interacting with, the user.



Each Widget is run inside a separate client process which provides a sandbox so that it doesn't affect any other Widgets or applications. For reliability, Dashboard also carefully manages Widgets. If a

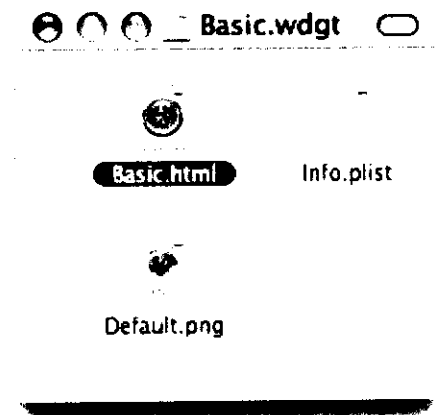
Widget crashes, it is automatically restarted so that it simply reappears in the Dashboard. If it misbehaves, crashing more than three times in a row, it is automatically removed from the Dashboard.

What's Inside a Widget?

At its simplest, a Widget is simply a web page that is displayed in the Dashboard rather than in a browser, such as Safari. A Widget is contained on disk in a bundle—a directory that groups all the needed resources for the Widget together in one place. Widget bundles are named with the `.wdgt` extension. Like any other bundle, a Widget's bundle is managed by the Finder as a single entity.

A basic Widget contains the following files:

- A main HTML file defining the user-interface for the Widget.
- A default background image in PNG format that can be displayed by Dashboard while it loads the Widget. The PNG format is used because of its excellent support for alpha channel transparency.
- A property list file named `Info.plist` that contains a Widget's identifier, name, version information, size, and the main HTML page as well as other optional information used by Dashboard.



As a Widget grows, you can place other files, such as images and external CSS and JavaScript files, into the Widget's bundle.

Let's take a look at the various components of a Widget, starting with HTML.

HTML Markup

As stated earlier, if you know how to write content for the web, you can write a Widget. To adhere to current web-standard best practices, it's recommended that you create content that complies with the XHTML 1.0 (Strict, Transitional, or Frameset) standard. For example, here's the HTML for a "Hello World" Widget:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Hello World</title>
</head>
<body background="Default.png">
<h1>Hello, World!</h1>
</body>
</html>
```

Since a Widget is defined using HTML, you can preview the Widget using Safari while writing the markup in your text editor of choice.

HTML provides most of the functionality needed for Widgets to be built, but a few extensions have also been added that give Widgets some unique capabilities you'll want to use. Specifically, there are three extensions to HTML used by Widgets. These are:

- A `canvas` tag allowing you to perform full 2D rendering in your Widget.
- A `composite` attribute on the `img` tag that lets you specify how an image is rendered.
- A new search type, `<input type="search">`, that allows you to create a Mac OS X-style search box.

It is important to note that these are a very small number of changes to the HTML specification and are designed to work in any application context where HTML could be used to create a user-interface. Apple is working with other vendors to ensure that these changes are adopted and become standardized.

Cascading Style Sheets

If you were to load the basic "Hello World" Widget above, you'll see that the "Hello World" text won't look acceptable. End users won't appreciate misplaced text. The solution is to use Cascading Style Sheets (CSS), the web standard for styling HTML.



Without CSS

With CSS

Here's an example of using CSS to style the text in the Widget:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<style>
body {
    margin: 0;
}

.helloText {
    font: 24px "Lucida Grande";
    font-weight: bold;
    color: white;
    position: absolute;
    top: 24px;
    left: 30px;
}
</style>
</head>
<body background="Default.png">
<h1 class="helloText">Hello, World!</h1>
</body>
</html>
```

Of course, it's also possible to put the CSS into a separate stylesheet file and refer to it using a `import` statement, as shown here:

```
<style>
    @import "HelloWorld.css";
</style>
```

As you can see, HTML and CSS give you a rich set of tools to create interesting user interfaces. Now, let's look at how to make those interfaces dynamic.

JavaScript

In keeping with the use of web technologies, Widgets use JavaScript to provide dynamic behavior. JavaScript in Dashboard works the same way as it does in any browser with the single addition of the `widget` object. The `widget` object gives you the ability to do the following:

- Access to the user preferences system.
- Perform transition, such as resizing the size of a Widget or flipping it over to access preferences.
- Respond to Dashboard activation events.
- Open other applications.

- Handle drag-and-drop operations.
- Retrieve content from the Internet using URLs.
- Execute system commands, such as shell scripts or command-line tools.

For example, to access preferences, you can use the following:

```
widget.setPreferenceForKey(preference, key);
widget.preferenceForKey(key);
```

Dashboard activation events are important for a Widget to handle so that it doesn't consume CPU time or network resources unless it is visible. The following example code shows how to do this:

```
if (window.widget) {
    widget.onhide = onhide;
    widget.onshow = onshow;
}

function onshow() {
    if (timerInterval == null) {
        timerInterval = setInterval("updateTime(true);", 1000);
    }
}

function onhide() {
    if (timerInterval != null) {
        clearInterval(timerInterval);
        timerInterval = null;
    }
}
```

As in any HTML page, you can attach JavaScript actions to any user interface element. For example, a calculator Widget can perform an action when a user clicks an equals image as follows:

```

```

An important note about the `widget` object: It is only available while running in Dashboard. It's not available when you run an object in Safari.

The Info.plist File

The Info.plist file contains the essential information about a Widget. This information is used by Dashboard to identify the Widget and where to find its resources. The easiest way to create a property list is to use the Property List Editor application, located in the `/Developer/Applications/Utilities`.

A few of the keys that can, or must, appear in a Widgets Info.plist file:

Key Name	Purpose	Required
MainHTML	The path in the Widget bundle to the main HTML file for the Widget's interface.	YES
Width	The width, in pixels, of the Widget.	YES
Height	The height, in pixels, of the Widget	YES
CFBundleName	The name of the bundle.	YES
CFBundleIdentifier	The reverse Internet domain style identifier for the bundle.	YES
DefaultImage	The path in the Widget bundle to the Widget's default image.	YES
Plugin	The name of a custom plug-in used by the Widget.	NO

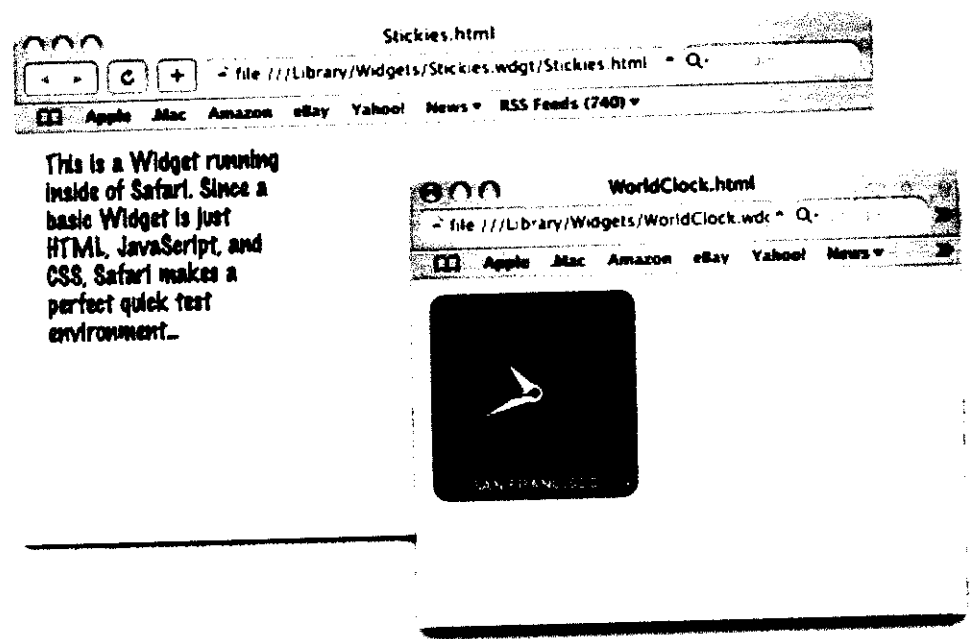
Creating a Widget

Creating a new Widget is easy and straightforward. Use the following process to get started:

- Create a directory to hold your Widget.
- Define the Widget's Info.plist file.
- Create the main HTML file using your favorite text editor.
- Open the HTML file in Safari to view the running Widget.
- Reload the Widget in Safari to see updates and changes that you have made.

This is a fast, easy, iterative approach familiar to legions of web developers. Of course, there's an even faster way to get started that will also be familiar to web developers: copy one of the working examples from the Dashboard SDK and use it as a starting point. (Note: The SDK is pre-release software and available only to ADC Select and Premier members.)

From the Widget bundle, you can enter straight into the test environment for Widgets: Safari. A simple double-click on the main HTML file opens it in Safari where you can interact with and reload it as you make changes to the Widget's contents.



You should keep in mind that Safari serves as a basic test environment for Widgets and it doesn't support all of the functionality available in the Dashboard. To test a Widget in the Dashboard, simply double-click on a .wdgt package. To see any changes, reload with Command-R. You'll know it has reloaded by the cool spinning effect that Dashboard performs on the Widget.

Leveraging Mac OS X Technologies

The capabilities of HTML, CSS, and JavaScript don't define the entire spectrum of what is possible in a Widget. In fact, they just mark the starting point. From there, you can access the deep capabilities of Mac OS X.

UNIX Commands

Any UNIX command or script, including those written in sh, tcsh, bash, tcl, Perl, or Ruby as well as AppleScript, can be accessed from the widget object. Here's an example of calling a UNIX command:

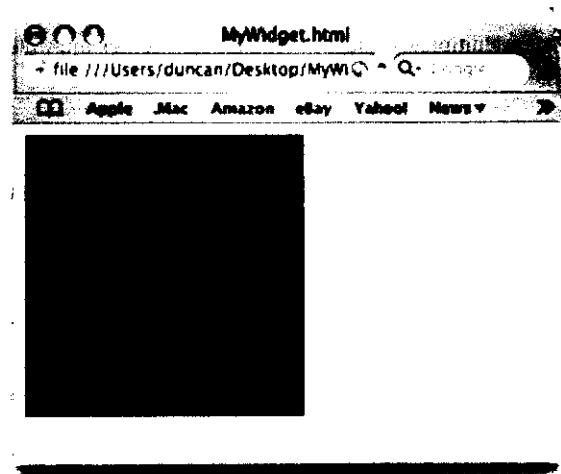
```
var obj = widget.system("ps -aux | grep Dashboard", null);
alert(obj.outputString);
```

This ability to tap into the command-line means that an amazing amount of power can be accessed from any Widget.

Quartz Drawing

Not only can Dashboard Widgets use HTML markup to display a user interface, they can also draw arbitrary graphics using a canvas tag. The drawing methods provided closely follow the capabilities of Quartz and provide the ability to composite images, draw antialiased lines, and even create complex quadratic-based paths. An example of this functionality is the World Clock Widget, which rotates and composites several images into the interface you see.

There are two steps in using a canvas in a Widget. The first is setting up an area in the HTML body to draw within, and then drawing into that canvas from JavaScript. The following simple example shows a canvas object being set up, then a blue square drawn, and then a red square drawn with transparency. The result of this code, shown to the right, is a purple square surrounded by a blue square.



```
<html>
<head>
</head>
<body>
<canvas id="canvas" width="200" height="200">
<script>
    var canvas = document.getElementById("canvas");
    var context = canvas.getContext("2d");
    context.clearRect(0, 0, 200, 200);
    context.setFillStyle(0.0, 0.0, 1.0, 1.0);
    context.fillRect(0, 0, 200, 200);
    context.setFillStyle(1.0, 0.0, 0.0, 0.5);
    context.fillRect(20, 20, 160, 160);
</script>
</body>
</html>
```

If you compare the simple calls here to those used to access Quartz 2D functionality from C or Objective-C code, you will recognize the form and naming of the calls.

Internet Plug-ins

Since Widgets are built using Web Kit, any Internet plug-in can be run from within the Widget. For example, a Widget could display a QuickTime-based movie using the QuickTime Internet plug-in. As well, a complex Shockwave-based application could be reused in the scope of a Widget.

Custom Code Plug-Ins

If you need to dig deeper into the system, or if you need to tap into your own application to create a Widget that closely interacts with it, you can create your own Cocoa-based plug-in. These plug-ins work by providing a JavaScript object that's made available to the Widget.

Widget plug-ins are created in Xcode using the "Cocoa Bundle" template. Once the project has been set up, the main class of the plug-in must implement the following methods:

```
- (id) initWithWebView:(WebView *) webView;
- (void) windowScriptObjectAvailable:(WindowScriptObject *) windowScriptObject;
```

The first method is called when your Widget first loads the plug-in. The second method is then called and allows you to provide a JavaScript object that your Widget can use. This bridges the gap between JavaScript and Objective-C. For example, the following implementation creates a JavaScript object with the name `MyScriptObject` and allows methods on that object to be called from JavaScript:

```
- (void) windowScriptObjectAvailable:(WindowScriptObject *) windowScriptObject
{
    MyScriptObject *myObj = [[MyScriptObject alloc] init];
    [windowScriptObject setValue:myObj forKey:@"MyScriptObject"];
}
```

The object can be accessed from JavaScript as follows:

```
<script>
  if (window.MyScriptObject) {
    MyScriptObject.someMethod(someArg);
  }
</script>
```

Using this plug-in model, you can expose most any kind of functionality from the system or from your application to a Widget.

Deploying a Widget

Once you have created a Widget, the next step is to get it into the hands of your users. For a Widget to be found by the Dashboard, it needs to be located in one of the following locations:

- /Library/Widgets
- ~/Library/Widgets

Conclusion

As you can see, Dashboard provides a great new development environment. Widgets are easy and fast to create, and yet very powerful. They have full access to the Internet, can use the advanced drawing capabilities of Quartz, and the rest of the functionality available in Mac OS X. The opportunities to be creative are enormous.

Get Started Now

Tiger doesn't ship until sometime in the first half of 2005. But as a developer, you can get started working with pre-release builds of Tiger now, and be ready to release your application when Tiger ships. How? Register with the Apple Developer Connection and take advantage of the Tiger Early Start Kit, which will give you everything you need to start your Tiger development today, including pre-release versions of Tiger and Xcode 2.0, and the Tiger ADC Reference Library.

Once you have Tiger, download the Dashboard SDK from the Apple Developer Connection, learn or leverage your experience with HTML, CSS, and JavaScript, and let your imagination run wild.

Reference Material

Since Widgets heavily utilize web technologies, some of the best reference material for building Widgets has already been published:

- JavaScript: The Definitive Guide, 4th Edition by David Flanagan (O'Reilly Media, 2001)
 - Cascading Style Sheets: The Definitive Guide, 2nd Edition by Eric Meyer (O'Reilly Media, 2004)
 - Dynamic HTML: The Definitive Reference, 2nd Edition by Danny Goodman (O'Reilly Media, 2002)
- Also, for detailed information about Web Kit's support for various web standards check the Safari Developer FAQ

Posted: 2004-12-07

Get information on Apple products.
Visit the Apple Store online or at retail locations.
1-800-MY-APPLE

Copyright © 2005 Apple Computer, Inc.
All rights reserved. | Terms of use | Privacy Notice